Modeling the MVM-Adapt System by Compositional I/O Abstract State Machines

Silvia Bonfanti[1][0000−0001−9679−4551], Elvinia Riccobene[2][0000−0002−1400−1026], and Davide Santandrea[2] Patrizia Scandurra[1][0000−0002−9209−3624]

[1] University of Bergamo, Bergamo, Italy
{silvia.bonfanti,patrizia.scandurra}@unibg.it
[2] Università degli Studi di Milano, Milan, Italy
elvinia.riccobene@,davide.santandrea@studenti.}unimi.it

Abstract. With the increasing complexity and scale of software-intensive systems, model-based system development requires composable system models and composition operators.
In line with such a vision, this paper describes our experience in modeling the behavior of the MVM-Adapt, an adaptive version of the Mechanical Ventilator Milano that has been designed, certified, and deployed during the COVID-19 pandemic for treating pneumonia. To keep the complexity of the requirements and models under control, we exploited a compositional modeling technique for discrete-event systems based on Abstract State Machines (ASMs). Essentially, separate ASMs represent the behavior of interacting subsystems of the MVM with their new adaptive functionalities; they can communicate with each other through I/O events, and co-operate by a precise orchestration schema.

Keywords: Compositional I/O Abstract State Machines · Discrete Event Systems modeling · ASMETA.

1 Introduction

With the increasing complexity, heterogeneity, and scale of software-intensive systems, model-based system development requires composable system models and the composition of their analysis [15]. Consequently, to design and reason about behavior and quality of a system it is necessary to develop separate and more manageable models of the system's subsystems/components, which can be first analyzed separately and then combined to analyze the overall behavior and quality of the system under development. In line with such a vision, in [9] we introduced a novel compositional modeling and simulation technique for discrete-event systems (DESs) based on the Abstract State Machine (ASMs) formal method [5, 10] and on typical workflow patterns such as parallel composition and cascading. Model-based simulation of (possibly distributed) DESs is an accepted practice for a reliable prototyping of their behavior, and sometimes the only alternative available in practice when systems are complex and

scalable [6, 16]. In [9] we introduce the concept of I/O ASM and their assembly (by suitable compositional operators) to model systems partitionable into distinct subsystems/components that interact for sharing resources in terms of input/output events. Each component can be modeled by an I/O ASM having its own input (monitored locations of the ASM), current state (controlled locations of the ASM), and output (out locations of the ASM). I/O ASMs interact in a black-box manner by suitably binding their inputs/outputs.

This paper provides a practical application of this compositional modeling technique [9] to a complex case study in the healthcare domain. Specifically, we present our experience in modeling *MVM-Adapt*, an adaptive version of the mechanical lung ventilator MVM [4] – Mechanical Ventilator Milano – that has been designed, certified, and deployed during the COVID-19 pandemic.

To keep the complexity of the requirements and models under control, we show how we managed the specification of the MVM-Adapt system as a composition of different ASM models, each representing the behavior of an independent and interacting subsystem with new adaptive functionalities; components can communicate with each other through I/O events, and co-operate by a precise orchestration schema. In particular, to model the MVM-Adapt system, we refined the ASM models [8] of the original MVM system by establishing precise I/O signal interfaces and adding the behavior of new adaptive ventilation. These ASM models were first validated, and verified separately, and then co-simulated (although these analysis results are not shown here for lack of space).

This paper is organized as follows. Section 2 presents the MVM-Adapt. Section 3 describes the I/O ASM models of the two main MVM-Adapt subsystems. Section 4 reports on related works. Section 5 concludes the paper.

2 Motivating example: MVM-Adapt

The MVM system [4] – Mechanical Ventilator Milano – was developed as part of an international research project during the COVID-19 pandemic with the goal of making up for the lack of mechanical ventilators in hospitals by quickly developing a prototype with low-cost components. Although the first MVM prototype was realized in less than one month, it required more than three extra months of full-time work of around 60 people (among them computer scientists and engineers) to go through the system development process to get the certification by the competent authority (FDA in the USA, CE in Europe). To give an idea of MVM complexity, its detailed behavior is described in about 1000 requirements sentences. The software controller has its own document of 31 pages and 157 requirements. Due to time constraints and lack of skills, no formal method was applied to the MVM project. Later, we assessed the feasibility of developing (part of) the ventilator by using formal methods [8] and a component-based formal specification development [9].

MVM was originally designed to provide two basic ventilation modes, the two most suitable to treat people with COVID-19 pneumonia: *Pressure Support Ventilation* (PSV) and *Pressure Control Ventilation* (PCV). In PSV mode, MVM

provides support to the patient that is partially unable to breathe on his/her own. In PCV mode, MVM controls the respiration cycle of the patient, who is completely unable to breathe on his/her own. However, the MVM project has grown and has changed into the *MVM-Adapt* project[3] with the goal of providing MVM with the *Adaptive Support Ventilation* (ASV) mode, as required by the most advanced mechanical ventilators [11]. In the *MVM-Adapt* project, we have implemented ASV as a user-selectable ventilation mode.

3 MVM Adapt: Architecture and Models

Beyond the HW subsystem, MVM-Adapt consists of three further components: GUI, controller, and supervisor. The *GUI* allows medical operators to set all the required parameters for ventilation and the alarm thresholds, and it displays all the data detected from the patient. The *controller* sets the hardware according to the *user* (medical staff) input, sets the inspiration and expiration valves on the base of the phase of the respiratory cycle, and notifies warning and alarms. The *supervisor* checks all the actions performed by the controller to assure patient safety (e.g., it checks the state of the valves and all the respiration parameters). In case of incorrect operation, it brings the system to a *fail-safe* state operating directly on the hardware (bypassing the controller). It is also responsible for controlling the ventilation change to adaptive mode in order to help the patient to use his/her own lungs as much as possible.

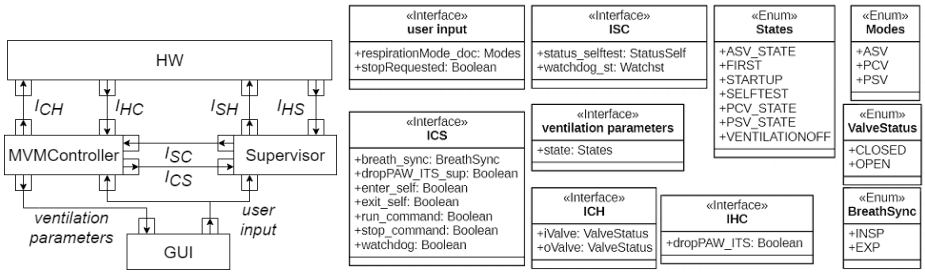
Fig. 1a provides a graphical view of the component assembly of the whole MVM ventilator (made of the components GUI, HW, MVMcontroller e Supervisor) and the bindings among all the component models as wires labeled with the name of I/O interfaces representing the binding functions (the exchanged signal values). The main I/O interfaces are shown in Fig. 1b using the UML notation.

We abstract here from modeling the GUI and the HW components, since they are not relevant for the adaptive feature of the MVM.

The MVM-Adapt system is the result of composing the two I/O ASMs, MVMcontrollerAdapt and Supervisor, by the *half-duplex bidirectional pipe* (<|>) [9] composition operator. The I/O ASM assembly can be expressed by the formula (MVMcontrollerAdapt <|> Supervisor).

This compositional modeling and simulation technique is supported by a specific tool, `AsmetaComp`, of the ASMETA [5] tool-set for ASMs. At each computational step of the assembly, according to the operational semantics of the composition operator <|>, `AsmetaComp` first executes the I/O ASM MVMcontrollerAdapt, then it uses the output of the MVMcontrollerAdapt as input to execute the Supervisor; the output of the Supervisor will be provided as input of the MVMcontrollerAdapt at the subsequent step The components MVMcontroller and Supervisor are modeled as I/O *control state* ASMs and are described in the following subsections. Artifacts concerning the validation and verification of each

[3] MVM-Adapt (Milano Ventilatore Meccanico Adaptive in the presence of uncertainty, FISR (Covid-19) project, funded in 2021.



(a) Component architecture (b) Component interfaces

Fig. 1: MVM-Adapt: I/O ASM assembly and interfaces

component and their compositional simulation to validate the reliability of the adaptive behavior of the MVM, are here skipped for the lack of space.

3.1 Adaptive Controller

The model of the controller for the adaptive version of the ventilator is an extension of that presented in [8, 9] for the basic MVM (with no adaptive features).

The I/O ASM MVMcontroller has input functions $I = I_{HC} \cup I_{SC} \cup user\_inputs$ and out functions $O = I_{CH} \cup I_{CS} \cup ventilation\_parameters$ (see Figure 1).

The set I is the union of the binding with:

- the GUI component providing the controller the *user inputs* – e.g., respirationMode_doc given by the doctor for changing the controller mode of operation (PCV, PSV or ASV), and stopRequested to stop the ventilation;
- the HW component (I_{HC}) – as the respiration parameters provided by sensors attached to the patient;
- the Supervisor component (I_{SC}) – as the alarm parameters and the signals watchdoc_st used by the Supervisor to communicate to the controller an alarm/(in)acting condition, and respirationMode_out used to communicate the change in ventilation mode.

The set O represents the bindings of the controller component with:

- the HW (I_{CH}): the out functions iValve and oValve are the input to the HW component to set the status of the input and output valves during ventilation;
- the Supervisor (I_{CS}): e.g., breath_sync, which indicates the current patient's inspiratory/expiratory phase; watchdog, which denotes alive communication between controller and supervisor; respirationMode_sup used to notify the current ventilation mode to the Supervisor; run_command and stop_command, which communicates that the ventilation has started or stopped;
- the GUI, which visualizes the current state of the controller.

The controller moves through six states as shown in Fig. 2; Code 1 reports, by using the `AsmetaL` textual modeling language, excerpts of the I/O ASM

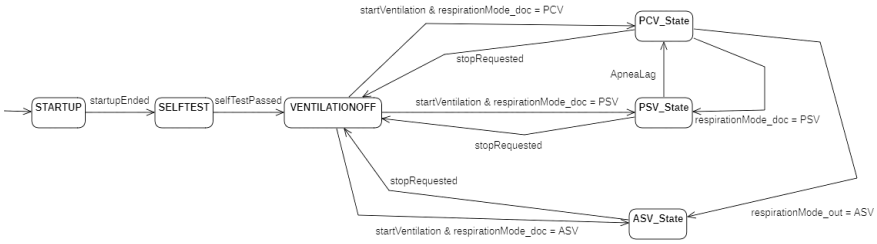


Fig. 2: MVM-Adapt: controller state machine

`MVMcontroller`[4]. The main rule shows the transition between states: at each step, depending on the current state value, a corresponding rule fires through the r_Main execution. The boolean monitored conditions labeling the transitions of the controller state machine in Fig. 2 and enabling the state change, are embedded, as rule guards, into the corresponding state change rule. E.g., the rule r_startup (see the right column in Code 1), that executes the state change from STARTUP to SELFTEST, is guarded by the monitored condition startupEnded, which yields true when the starting phase is completed and the ventilator parameters have been initialized with default values.

In the initial state STARTUP, the controller sends the signal watchdog to the supervisor in order to establish communication with it. Moving from state STARTUP to SELFTEST, a signal (by the out function enter_self) (line 13 in Code 1) is sent to the supervisor to notify it that the self-test phase has been started. In SELFTEST state, the controller performs a sequence of tests ensuring that the hardware is fully functional. If this phase ends with a positive outcome (selfTestPassed is true), the controller notifies the supervisor that the self-tests have been completed (by the out function exit_self) and moves to VENTILATIONOFF. In this state, the ventilator does not operate, and the valves are put in a safe position (the input valve is closed and the output valve is opened). When the patient is ready for ventilation (i.e., startVentilation is true), the physician selects PCV, PSV, or ASV mode (by respirationMode_doc input function), and the controller moves to the corresponding state (PCV_STATE, PSV_STATE or ASV_STATE) upon notifying the supervisor that the ventilation has started (by out function run_command). When ventilating, the ventilation mode can be changed, manually or automatically, and the controller changes its state accordingly. The transition from PCV to PSV is set by the physician; the transition from PSV to PCV occurs when the patient is not able to breathe on his/her own and he/she remains in apnea for a certain period of time. The ventilation continues until the physician requests stopRequested; in this case, the controller returns to VENTILATIONOFF and notifies the change to the supervisor by the out function stop_command.

[4] All models and analysis artifacts are available in `https://github.com/asmeta/asmeta_based_applications/tree/main/MVM/MVM%20Cosimulation%20ABZ2023`

```
1  asm MVMcontroller
2  signature:
3   enum domain States = {STARTUP |
4     SELFTEST | VENTILATIONOFF |
5     PCV_STATE | PSV_STATE | ASV_STATE}
6  enum domain Modes = {PCV | PSV | ASV}
7     ...
8
9  main rule r_Main =
10 par
11    if state = STARTUP then  r_startup[]  endif
12    if state = SELFTEST then r_selftest[] endif
13    if state = VENTILATIONOFF then
14       r_ventilationoff[] endif
15    if state = PCV_STATE then r_runPCV[] endif
16    if state = PSV_STATE then r_runPSV[] endif
17    if state = ASV_STATE then r_runASV[] endif
18 endpar
```

```
controlled state: States
monitored respirationMode_doc: Modes
monitored respirationMode_out: Modes
monitored stopRequested: Boolean
out iValve: ValveStatus
out oValve: ValveStatus
out respirationMode_sup: Modes
   ...
   rule r_startup =
      if startupEnded then
       par
       state := SELFTEST
      enter_self := true
     endpar
      endif

 default init s0:
      function state = STARTUP
```

Code 1: MVMController model in the ASMETA textual notation

The controller can move to state ASV_STATE in two ways (see Fig. 2) when the physician sets this ventilation mode for the patient by the GUI (signal respirationMode_doc), or if the supervisor determines to change the ventilation mode from PCV to ASV (notified by the input function respirationMode_out) on the base of the ventilation parameters.The rule r_runASV is responsible for managing the ASV ventilation mode. When the ventilator operates in ASV mode, the controller performs the following actions: it sets the in and out valves to allow the patient's inspiration (resp. expiration), resets the timers to compute the duration of the next respiratory (insp/exp) phases, computes the target ventilation parameters – the target volume of area to be inspired/expired and the target respiratory rate – by suitable equations[5] that guarantee safe ventilation for the patient according to the Otis curve [13], and communicates the current patient's respiration mode to the supervisor (by out function breath_sync).

3.2 Adaptive Supervisor

The I/O ASM Supervisor goes through a sequence of six states as shown in Figure 3. Some states correspond to those in the controller machine, since they reflect the configuration of both components during the operation of the MVM-Adapt, as the stating phase, the self-test, and the off ventilation. State transitions are driven by the main rule shown in Code 2.

The supervisor starts in state STARTUP, when the ventilator is turned on and the parameters initialized with default values. When the signal watchdog is received from the controller, it moves to the INIT state (by the rule r_startup, see line 11 in Code 2), in which the supervisor performs the following checks

[5] Note that these complex formulas have been modeled, but they are not shown here to keep simple the presentation of the case study.

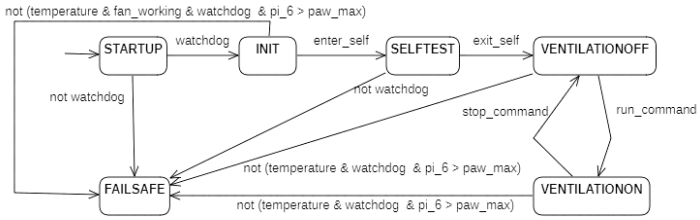


Fig. 3: MVM-Adapt: supervisor state machine

```
1   main rule r_main =
2    par
3     if state = SELFTEST then  r_selftest[] endif
4     if (state != SELFTEST and state != FAILSAFE) then
5      par
6       r_check_adc[]
7       r_check_pi6[]
8        if (adc_reply = RESPONSE) and (pi_6_reply = RESPONSE) then
9            if (fan_working) then
10            par
11             if state = STARTUP then r_startup[]  endif
12             if state = INIT then r_init[] endif
13             if state = VENTILATIONOFF then r_ventilation_off[] endif
14             if state = VENTILATIONON then r_ventilation_on[] endif
15            endpar
16           else r_failsafe  endif
17     endif endpar endif  endpar
```

Code 2: Adaptive Supervisor – main rule

by means of the r_init rule: the device temperature, the pressure level, the fan operation, and the communication with the controller (by the watchdog signal). If there are no errors or inconsistencies with the expected values, and it receives from the controller information that it has successfully ended the startup phase and it has started the self-test phase (by the signal enter_self), the supervisor moves to the SELFTEST state and starts a sequence of tests on the HW parts (by executing the rule r_selftest, line 3 in Code 2). When it receives from the controller information that it has (successfully) ended the self-test phase (by the signal exit_self), the supervisor moves to state VENTILATIONOFF, waiting for starting ventilation (see line 13). In this state, the supervisor checks if the temperature and the pressure levels are within the allowed ranges and that the communication with the controller is active. Once the run_command is received from the controller, the supervisor moves to the state VENTILATIONON. When the patient is being ventilated, by executing the rule r_ventilation_on (see line 14), the supervisor is responsible for managing alarms and for checking if ventilation parameters are within the set thresholds. Moreover, if a stop_ command is received from the controller, the supervisor returns to state VENTILATIONOFF.

If HW problems are revealed when the supervisor is in states INIT or VENTILATIONOFF/ON, or communication problems with the controller occur when it

is in one of the states STARTUP, INIT, SELFTEST, VENTILATIONOFF/ON, by executing the rule r_failsafe (here not reported), the supervisor moves to the state FAILSAFE and the ventilator is put into a safe configuration (in-valve closed, out-valve open, and alarm rising) in order to avoid patient harm.

The adaptive supervisor is also responsible for communicating (by means of the out function respirationMode_out) the ventilation mode change from PCV to ASV to the controller, in order to help the patient to use his/her own lungs as much as possible. This feature is not further described here.

4 Related Work

Existing approaches that inspired us are those related to workflow modeling and service orchestration (such as tools for the Business Process Model and Notation (BPMN) [2], and the Jolie language [1]), and to multi-state machine modeling (like Yakindu statecharts tools [3]). However, our approach is much more oriented to distributed model-based system simulation, useful, for example, in practical contexts where models have to be co-executed in tandem with real systems, such as runtime models that are part of the knowledge base of self-adaptive and autonomous systems [7] or of a *digital twin* plant [12].

Related to component- and service- based architectures, ASMs have been used for service behavior modeling and prototyping, in conjunction with the OASIS/OSOA standard Service Component Architecture (SCA) for heterogeneous service assembly. SCA-ASM compont implementations can be co-executed *in place* with other component implementations [14] and their reliability, both at system-level and component-level, can be calculated [14].

5 Conclusion

In this paper, we have shown how to use the concept of compositional I/O ASMs to model the MVM-Adapt case study. The technique is supported by the ASMETA tool `AsmetaComp`, which is intended for allowing distributed co-simulation of separate ASM system models.

From our modeling experience we have learned some relevant lessons: (i) the advantage of managing requirements complexity by dividing a system model into sub-models; (ii) the flexibility in managing the separation of the modeling tasks among different groups; (iii) the necessity to clarify the system requirements concerning component interfaces and communication protocols to establish a correct architecture and a component coordination schema; (iv) the easiness of adding, by model refinement, adaptive features to a system model already decomposed into sub-models.

In the future, we plan to evaluate the usability of the technique, to enrich the set of composition operators and to move toward the definition and implementation of choreography constructs to deploy and enact a choreography-based execution of asynchronous I/O ASMs.

References

1. Jolie, https://jolie-lang.org
2. OMG Business Process Model and Notation, https://bpmn.org/
3. YAKINDU Statechart Tools, https://itemis.com/en/yakindu/state-machine
4. Abba, A., et al.: The novel Mechanical Ventilator Milano for the COVID-19 pandemic. Physics of Fluids **33**(3), 037122 (mar 2021)
5. Arcaini, P., Bombarda, A., Bonfanti, S., Gargantini, A., Riccobene, E., Scandurra, P.: The ASMETA approach to safety assurance of software systems. In: Raschke, A., Riccobene, E., Schewe, K. (eds.) Logic, Computation and Rigorous Methods - Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday. Lecture Notes in Computer Science, vol. 12750, pp. 215–238. Springer (2021). https://doi.org/10.1007/978-3-030-76020-5_13, https://doi.org/10.1007/978-3-030-76020-5_13
6. Bañares, J.Á., Colom, J.M.: Model and simulation engines for distributed simulation of discrete event systems. In: Economics of Grids, Clouds, Systems, and Services. pp. 77–91. Springer International Publishing, Cham (2019)
7. Bencomo, N., Götz, S., Song, H.: Models@run.time: a guided tour of the state of the art and research challenges. Software and Systems Modeling **18**(5) (2019)
8. Bombarda, A., Bonfanti, S., Gargantini, A., Riccobene, E.: Developing a prototype of a mechanical ventilator controller from requirements to code with ASMETA. Electronic Proceedings in Theoretical Computer Science **349**, 13–29 (Nov 2021). https://doi.org/10.4204/eptcs.349.2
9. Bonfanti, S., Gargantini, A., Riccobene, E., Scandurra, P.: Compositional simulation of Abstract State Machines for safety critical systems. In: Tarifa, S.L.T., Proença, J. (eds.) Formal Aspects of Component Software - 18th International Conference, FACS 2022, November 10-11, 2022, Proceedings. LNCS, vol. 13712, pp. 3–19. Springer (2022). https://doi.org/10.1007/978-3-031-20872-0_1
10. Börger, E., Raschke, A.: Modeling Companion for Software Practitioners. Springer, Berlin, Heidelberg (2018)
11. Fernández, J., Miguelena, D., Mulett, H., Godoy, J., Martinón-Torres, F.: Adaptive support ventilation: State of the art review. Indian Journal of Critical Care Medicine **17**(1), 16–22 (2013). https://doi.org/10.4103/0972-5229.112149
12. Grieves, M.: Origins of the Digital Twin Concept (08 2016)
13. Hamilton Medical: Operator's manual 610862/05 3.44d2015-09-24, https://www.hamilton-medical.com/dam/jcr:64cf14ea-0659-40f4-809d-9882342ce206/GALILEO-ops-manual-en-610862.05.pdf
14. Riccobene, E., Scandurra, P.: A formal framework for service modeling and prototyping. Formal Aspects Comput. **26**(6) (2014)
15. Talcott, C., Ananieva, S., Bae, K., Combemale, B., Heinrich, R., Hills, M., Khakpour, N., Reussner, R., Rumpe, B., Scandurra, P., Vangheluwe, H.: Composition of Languages, Models, and Analyses, pp. 45–70. Springer (2021)
16. Weyns, D., Iftikhar, M.U.: Model-Based Simulation at Runtime for Self-Adaptive Systems. In: Kounev, S., Giese, H., Liu, J. (eds.) 2016 IEEE International Conference on Autonomic Computing, ICAC 2016. IEEE Computer Society (2016)